

Program Reasoning

12. Search Space Prioritization

Kihong Heo



Naive Enumerative Search

- Explore the search space in increasing size of programs (i.e., Occam's razor)
- With search space pruning techniques
- But, is this enough?

iter 0	x	y			
iter 1	$x + x$	$x - x$	$x + y$	\dots	$x \leq y$ \dots
iter 2	$x + x + y$	$x + x - y$	\dots	$\text{if } (x \leq y) \ y \ x$	\dots
iter 3	$x + x + x + y$	\dots	$\text{if } (x \leq y) \ (y + x) \ x$		

Problem of Enumerative Search

- Blindly search over the large search space without any guidance
- Two major problems:
 - Scalability: #programs grows exponentially in program size
 - Quality: may overfit the I/O examples
- For example, $f(-1,0) = 0 \wedge f(0,-1) = 0$

iter 0	x	y			
iter 1	$x + x$	$x - x$	$x + y$...	$x \leq y$...
iter 2	$x + x + y$	$x + x - y$...	$\text{if } (x \leq y) \ y \ x$...
iter 3	$x + x + x + y$...	$\text{if } (x \leq y) \ (y + x) \ x$		

But which one is more likely to be a solution?

$x - x$ vs. $\text{if } (x \leq y) \ y \ x$



Statistical Regularities in Programs

- Programs often contain **repetitive** and **predictable** patterns

`for (i = 0; i < 100; ??)`

- Statistical** program models: probabilistic distribution over programs
 - E.g., n-gram, probabilistic context-free grammar (PCFG), etc

$\Pr(?? \rightarrow i++ \mid \text{for } (i = 0; i < 100; ??)) = 0.85$

$\Pr(?? \rightarrow i-- \mid \text{for } (i = 0; i < 100; ??)) = 0.01$

- Applications: code completion, deobfuscation, program repair, etc

A Solution: Euphony*

- Enumerate programs by **likelihood**, not by program size
- Likelihood is provided by **probabilistic models** over programs
 - “*How likely is the candidate program?*”
- Try the **most likely** (highest probability) candidate first

*W. Lee, K. Heo, R. Alur, M. Naik., Accelerating Search-Based Synthesis Using Learned Probabilistic Models, PLDI, 2018

Probabilistic Language Model

- A probability distribution over sentences in a language
 - Conditional probability and the chain rule: $\Pr(w_1, w_2, \dots, w_n) = \prod_{i=1}^n \Pr(w_i \mid w_1, w_2, \dots, w_{i-1})$
 - E.g., $\Pr(I \text{ am a boy}) = \Pr(I) \times \Pr(am \mid I) \times \Pr(a \mid I \text{ am}) \times \Pr(boy \mid I \text{ am a})$
- Learned from a large corpus of sentences

Which program is more likely?

“I am a boy”

92%

“You am a boy”

1%

Which word is more likely to be next?

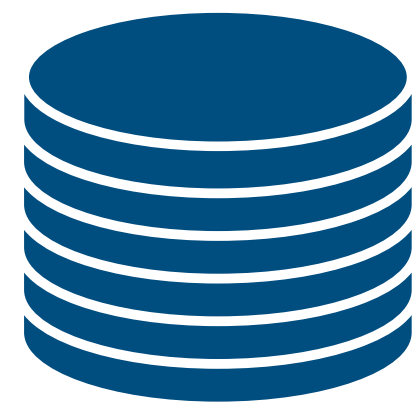
“I am a KAIST _____”

“I am a KAIST stupid.”

student ✕

A Naive Language Model

- $\Pr(w \mid h)$: the probability of a word w given some history h
 - E.g., $\Pr(\text{the} \mid \text{its water is so transparent that})$
- How to compute $\Pr(w \mid h)$?
- A naive idea: count



Large corpus
(e.g., Web)



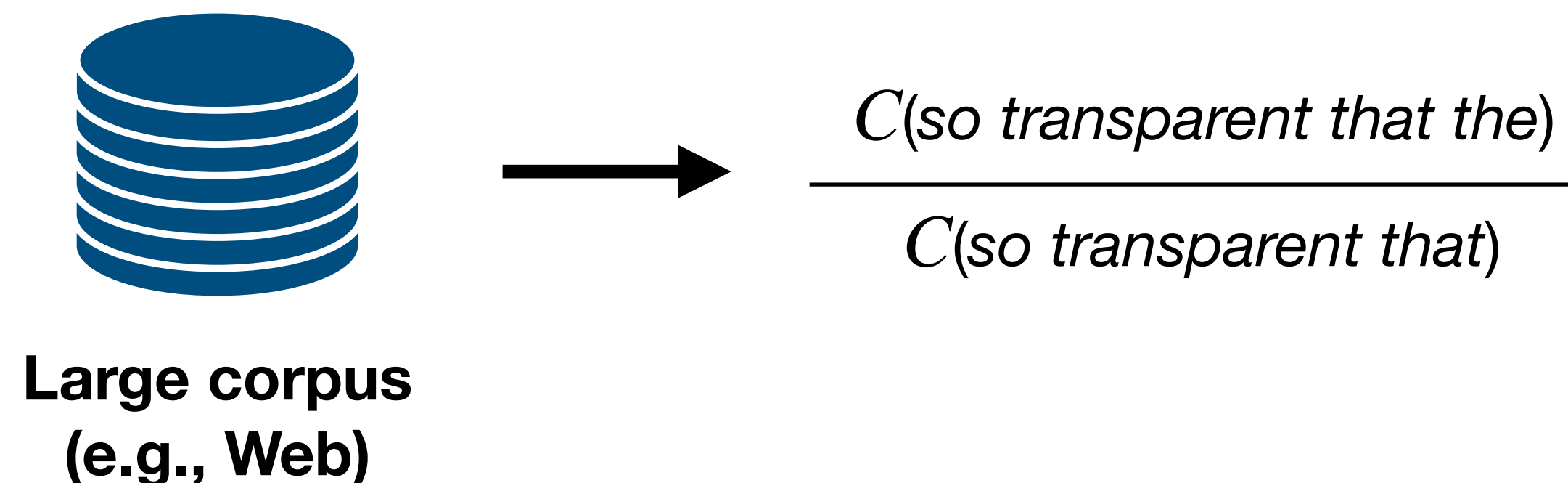
$$\frac{C(\text{its water is so transparent that the})}{C(\text{its water is so transparent that})} = \Pr(\text{the} \mid \text{its water is so transparent that})$$

Problem: Sparsity

- What is the probability of the following sentence S ?
 - S : “*KAIST’s duck pond water is so transparent that the*”
 - $\Pr(S) = \Pr(KAIST's) \times \Pr(duck \mid KAIST's) \times \dots$
 $\times \Pr(the \mid KAIST's duck pond water is so transparent that)$
 - $C(KAIST's duck pond water is so transparent that)$ over the web?
- Language is creative: new sentences are created all the time!
- Longer sentence: lower probability

N-gram

- Idea: approximate the history by the last N words
 - Unigram, bigram, trigram, 4-grams, ...
- Example: trigram
 - $\Pr(\textit{the} \mid \textit{KAIST's duck pond water is so transparent that}) \approx \Pr(\textit{the} \mid \textit{transparent that})$



Probabilistic Language Model for Programs

- A probability distribution over programs in a language
- Learned from a large corpus of programs (e.g., Github, Stackoverflow, etc)
- Example: N-gram, PCFG, PHOG, etc

Which program is more likely?

`"x - x"` `"if (x ≤ y) y x"`

1%

95%

Which word is more likely to be next?

`"for (i = 0; i < 100; ____)"`

`"for (i = 0; i < 100; j++)"`

i++ ×

Probabilistic Grammar (1)

- Idea: given a context, provide the probability of each production rule: $\text{Pr}(\text{rule} \mid \text{context})$
 - Context (history): sentential form $\in (N \cup \Sigma)^*$
- Ultimately assign a probability to each program
- Example

CFG $S \rightarrow x \mid 1 \mid S + S$

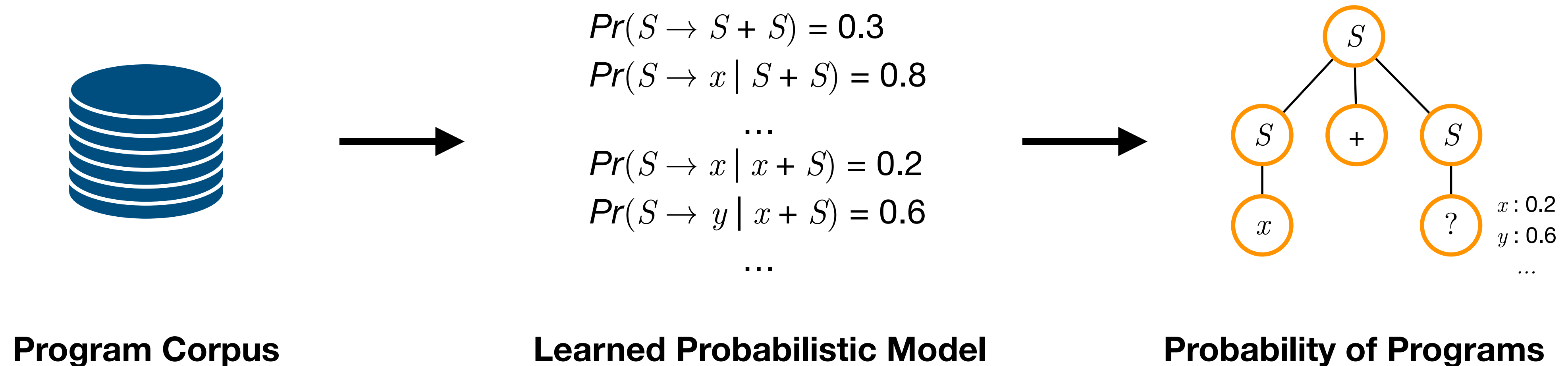
Probability of “x + 1”

$S \rightarrow S + S \rightarrow x + S \rightarrow x + 1$

$$\text{Pr}(x + 1) = \text{Pr}(S \rightarrow S + S \mid S) \times \text{Pr}(S \rightarrow x + S \mid S + S) \times \text{Pr}(S \rightarrow x + 1 \mid x + S)$$

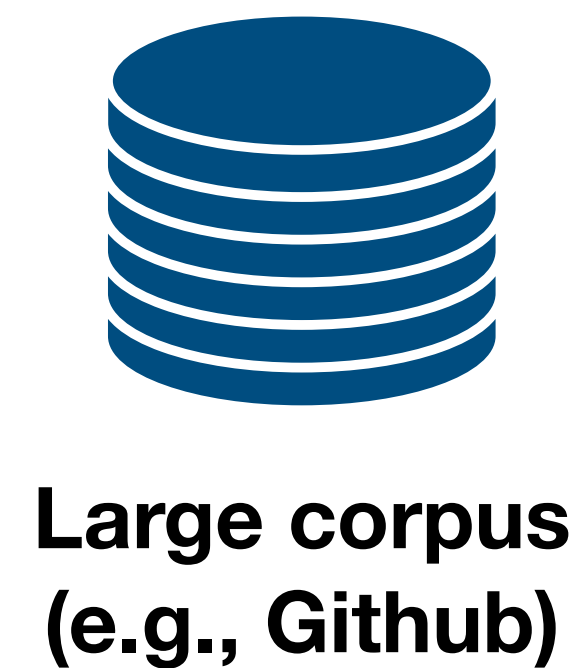
Probabilistic Grammar (2)

- Learn a probabilistic model of programs from a corpus of programs
 - Human-written or auto-generated programs by other synthesizers



PCFG

- Probabilistic Context Free Grammar (PCFG)
- One of the simplest form of probabilistic language model: ignore context
- Provide a probability to each production rule



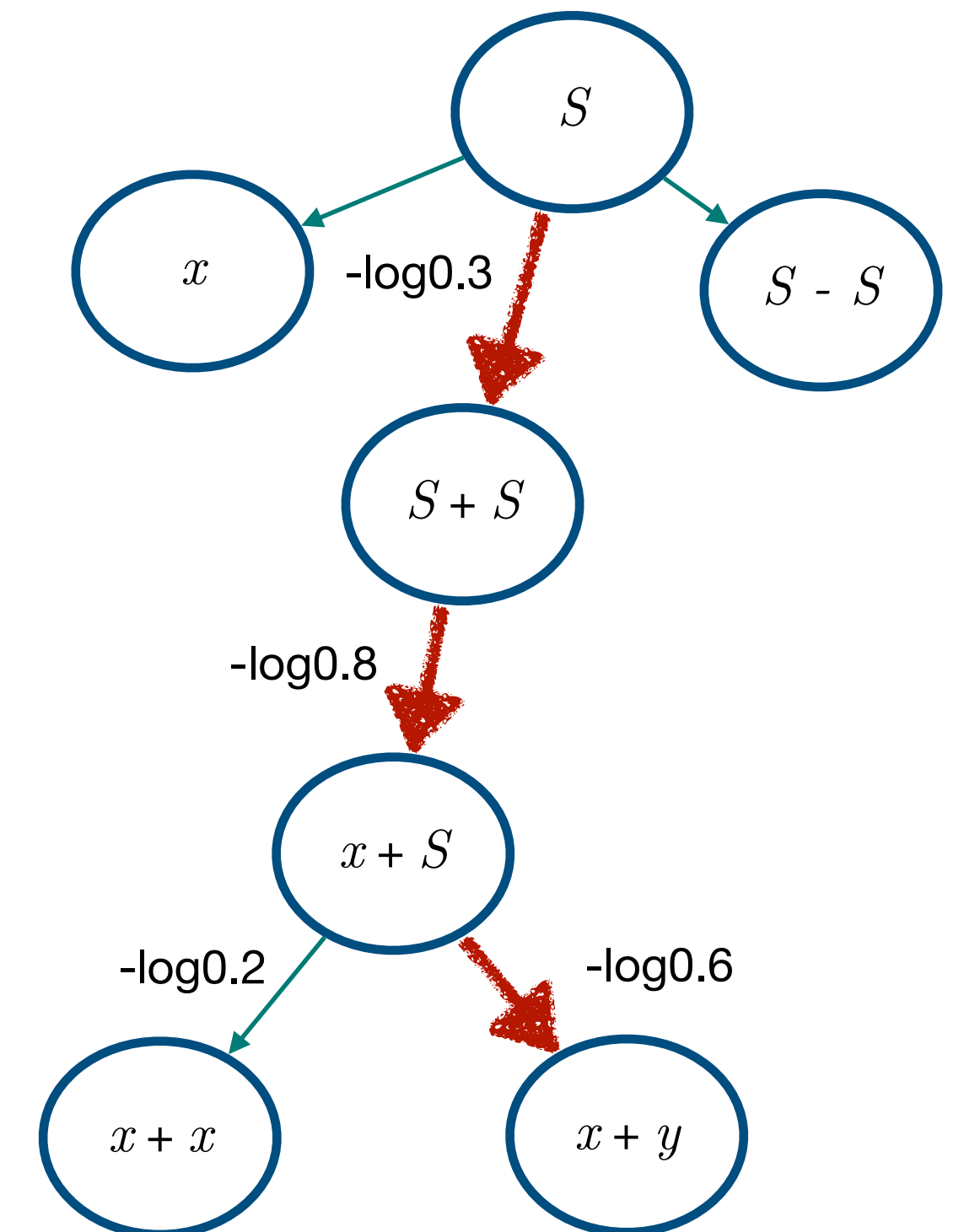
$$\xrightarrow{\frac{C(S \rightarrow t)}{C(S \rightarrow *)}} \xrightarrow{\quad}$$

$A \rightarrow \beta$	P
$S \rightarrow 0$	0.1
$S \rightarrow 1$	0.2
$S \rightarrow x$	0.3
$S \rightarrow S + S$	0.3
$S \rightarrow S - S$	0.1

Guided Enumeration by Probabilistic Model

- Given a model, construct a directed graph
 - Node: sentential forms
 - Weight: negative log probability of a production rule
- Compute the shortest path
 - starting from the start symbol to the program
 - E.g., Dijkstra's, A*, etc
- Enumeration by likelihood

$$\begin{aligned}
 \Pr(S \rightarrow x) &= 0.3 \\
 \Pr(S \rightarrow y) &= 0.2 \\
 \Pr(S \rightarrow z) &= 0.01 \\
 \Pr(S \rightarrow S + S) &= 0.4 \\
 \Pr(S \rightarrow S - S) &= 0.09
 \end{aligned}$$



$$x : 0.3 \quad y : 0.2 \quad x + x : 0.4 \times 0.3 \times 0.3 = 0.036 \quad x + y : 0.4 \times 0.3 \times 0.2 = 0.024 \quad z : 0.01$$

Guided Top-down Enumeration

```
top-down( $G = \langle \Sigma, N, R, S \rangle, \phi$ ):  
   $Q := \{(S, 0)\}$   
  while  $Q \neq \{\}$ :  
     $(p, d) := \text{dequeue\_min}(Q)$   
    if  $\text{ground}(p) \wedge \phi(p)$ : return  $p$   
     $P' := \text{unroll}(G, p, d)$   
    forall  $p' \in P'$ :  
      if not  $\text{equiv}(p, p')$ :  
        enqueue( $Q, p'$ )
```

```
unroll( $G = \langle \Sigma, N, R, S \rangle, p, d$ ):  
   $Q' := \{\}$   
   $A := \text{left-most non-terminal in } p$   
  forall  $(A \rightarrow B) \text{ in } R$ :  
     $p' := p[B/A]$   
     $Q' := Q' \cup \{(p', d + w(p, p'))\}$   
  return  $Q'$ 
```

Experimental Setup

- 1167 tasks from 3 different domains

	A	B	C
1	First Name	Last Name	Full Name
2	Kihong Heo	Kihong	Heo
3	Michael Jordan	Michael	
4	Thierry Henry	Thierry	
5			
6			

STRING: End-user programming for string manipulations
(205 tasks)

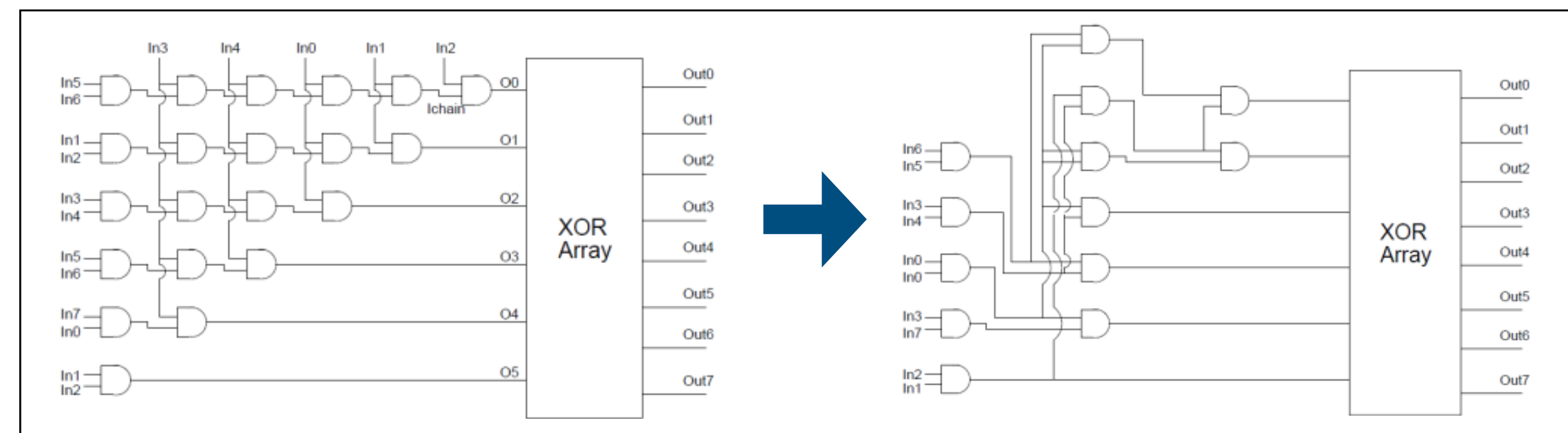
```
complement
~ 01010001110101110000000000001111
  1010111000101000111111111110000

bitwise and
01010001110101110000000000001111
& 00110001011011100011000101101110
  00010001010001100000000000001110

bitwise or
01010001110101110000000000001111
| 00110001011011100011000101101110
  011100011111111110011000101101111

bitwise xor
01010001110101110000000000001111
^ 00110001011011100011000101101110
  01100000101110010011000101100001
```

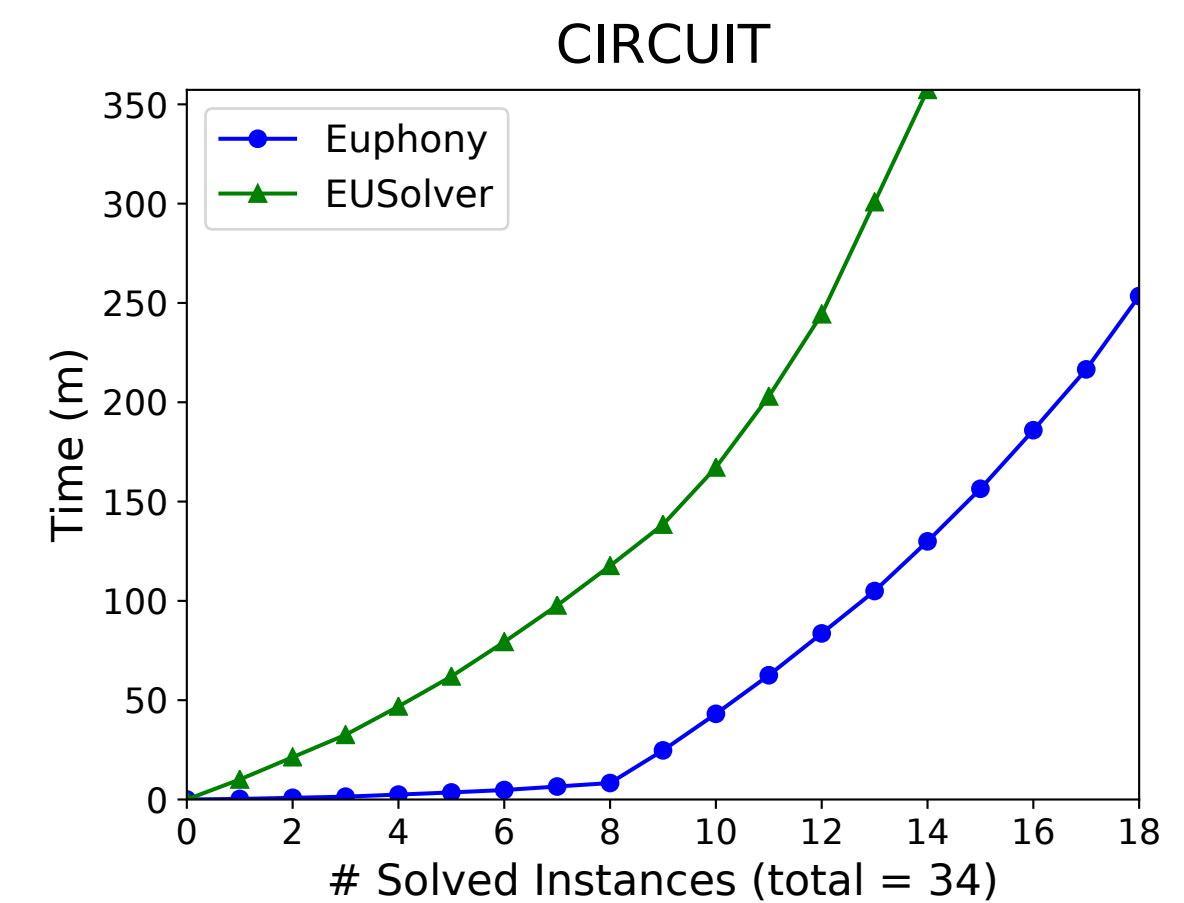
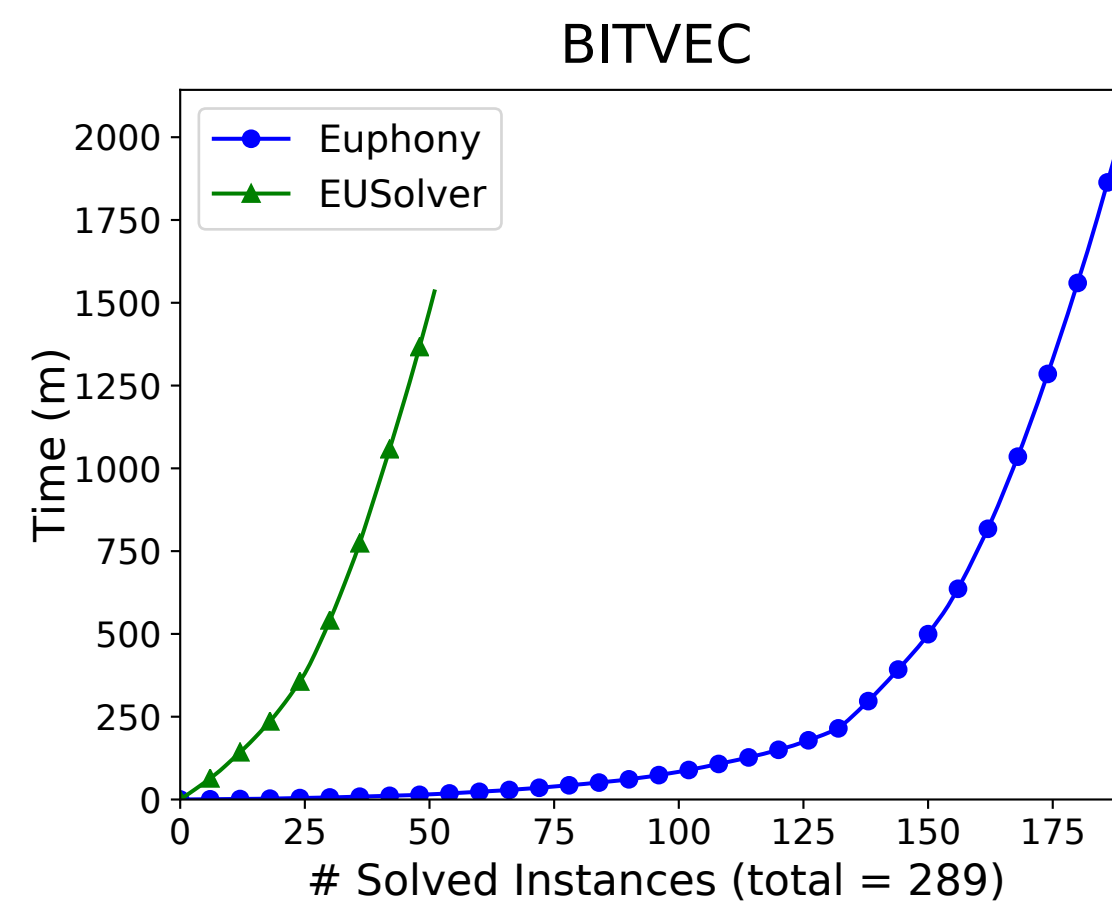
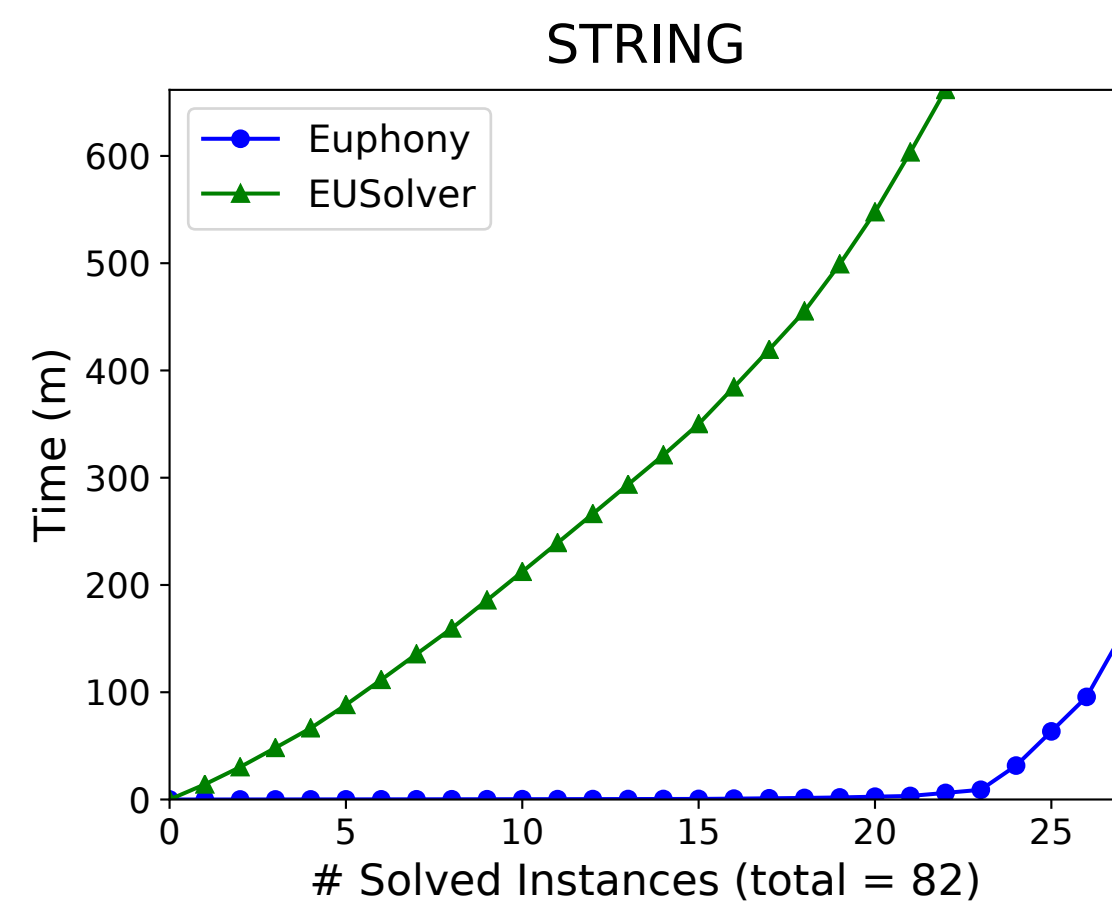
BITVEC: Efficient low-level algorithms
(750 tasks)



CIRCUIT: Attack-resistant crypto circuits generations
(212 tasks)

Effectiveness

- Comparison to EUSolver (a program synthesizer without prob. guidance)
 - Training: 762 tasks solved by EUSolver in 10 minutes
 - Testing: 405 (timeout: 1 hour)



Summary

- Problem: scalability and quality
- Euphony: a program synthesizer guided by a **learned probabilistic model**
 - E.g., probabilistic program model + shortest pathfinding
- Need a lot more research on efficient search
 - E.g., advanced learning techniques, static analysis, constraint solving, etc